

Za radoznale: Ponavljanje i sve je objekat

Ovo nije obavezno i neće biti na ispitu. Za one koji žele da vide kako se ista stvar radi naprednije. Svi ispisi su dobijeni pokretanjem u pygame-ce.

1. Zvezdica pri dodeli i dve zvezdice za rečnik

Na času je zvezdica raspakivala kolekciju u pozivu funkcije print. Zvezdica radi i pri dodeli: jedno ime dobije prvi element, a ime sa zvezdicom listu svih ostalih. Dve zvezdice raspakuju rečnik u imenovane argumente: ključevi postaju imena parametara.

```
prvi, *ostali = [10, 20, 30, 40]
print(prvi, ostali)

*bez_poslednjeg, poslednji = "sneg"
print(bez_poslednjeg, poslednji)

def pahuljica(x, y, velicina=3, boja="bela"):
    return f"{boja} pahuljica {velicina} px u ({x}, {y})"

podaci = {"x": 100, "y": 40, "boja": "plava"}
print(pahuljica(**podaci))           # isto kao pahuljica(x=100, y=40, boja="plava")
```

Ispis:

```
10 [20, 30, 40]
['s', 'n', 'e'] g
plava pahuljica 3 px u (100, 40)
```

Ime sa zvezdicom uvek dobije listu, čak i kad raspakujemo string. Poziv pahuljica(**podaci) je isto što i poziv sa tri imenovana argumenta; velicina zadržava podrazumevanu vrednost 3.

Zašto ovo nismo radili na času: na času nam je bilo dovoljno raspakivanje u print; ovo se koristi kad podatke o objektu čuvamo u rečniku, na primer kad ih čitamo iz fajla sa podešavanjima igre.

Fajl: `primeri/za_radoznale/01_r1_zvezdica_pri_dodeli.py`

2. Lista napravljena u jednom redu

U vežbi Level 2 smo novu listu pravili petljom i metodom append, a uz join smo koristili map. Python ima i kraći zapis koji radi isto: u uglastim zagradama napišemo šta je element i kroz šta prolazimo. Ovaj zapis se na engleskom zove list comprehension.

```
pahuljice = [(60, 40, 3), (150, 90, 5), (260, 30, 2)]
nove = [(x, y + 10, r) for (x, y, r) in pahuljice]
print(nove)

brojevi = [2026, 10, 6]
print("-".join([str(b) for b in brojevi]))

velike = [r for (x, y, r) in pahuljice if r >= 3]
print(velike)
```

Ispis:

```
[(60, 50, 3), (150, 100, 5), (260, 40, 2)]
2026-10-6
[3, 5]
```

Prvi red pravi istu listu kao petlja iz rešenja vežbe Level 2. Drugi primer daje isti rezultat kao join sa map(str, ...). Treći primer dodaje uslov: u listu ulaze samo poluprečnici od bar 3 piksela.

Zašto ovo nismo radili na času: petlja sa append je duža, ali se svaki korak vidi; kraći zapis je zgodan kad ga već dobro razumete.

Fajl: `primeri/za_radoznale/01_r2_lista_u_jednom_redu.py`

3. Zašto ključ rečnika ne može da bude lista

Nepromenljivost torke nije samo ograničenje: zbog nje torke sme da bude ključ rečnika ili element skupa, a lista ne sme. Python ključ pretvara u broj (heš) da bi ga brzo našao; kada bi se ključ promenio, rečnik ga više ne bi našao.

```
polja = {(0, 0): "jelka", (3, 1): "čikica"}
print(polja[(3, 1)])

posecena = {(0, 0), (1, 0), (0, 0)}
print(len(posecena))

try:
    lose = {[0, 0]: "jelka"}
except TypeError as greska:
    print("Greška:", greska)
```

Ispis:

```
čikica
2
Greška: unhashable type: 'list'
```

Torke (0, 0) i (3, 1) su ključevi, kao koordinate polja na tabli igre. Skup posećenih polja čuva (0, 0) samo jednom. Lista kao ključ daje TypeError: unhashable type: 'list'; try i except samo hvataju grešku da bi program nastavio i ispisao je.

Zašto ovo nismo radili na času: u našoj igri položaje čuvamo kao vektore (tema 07); torke kao ključ je korisna za igre na tabli i mreži, kao što su šah ili lavirint.

Fajl: `primeri/za_radoznale/01_r3_torka_kao_kljuc.py`